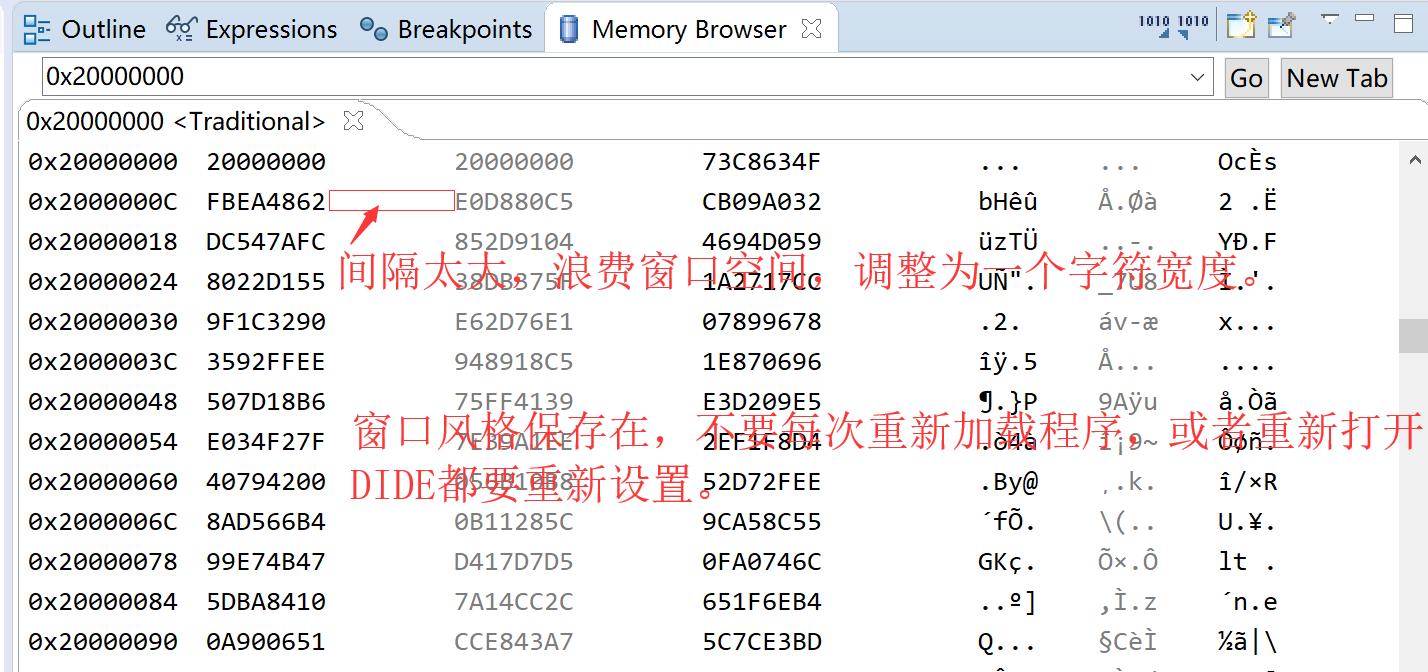
# 功能优化

1. memory窗口设置。



1. config一致性检查

源码随时可能修改配置部分（增加、修改、删除、移动、改属性），因此打开工程时，或者查看工程属性时，应该检查工程配置与源码中配置内容的一致性，要提出警告，并要求用户重新设置。方法如下：

1. 没有修改过的配置项，维持不变。
2. 删除的配置项，从project\_config.h中删除。
3. 被修改、增加的配置项，从源码中重新读取默认值，并在设置界面中用红色字体显示。
4. DIDE不能拖放到任务栏
5. 去掉PrjInit.c中Heap\_DynamicModuleInit后面的两行printf
6. 编译库的打包参数改为-rPu
7. 解决编译时总是停留在12%的问题。Libos丢失的问题

# shell实现

C的链接器工作时，对于.a库中的.o文件，如果该文件没有一个符号被引用，该文件就会被整体排除掉，如果其中有shell命令和变量，也会被排除掉，即使在lds中使用KEEP也无济于事，这是我们所不希望发生的。真实需求是，如果某组件被裁剪掉了，自然组件内定义的shell命令也灰飞烟灭，无可厚非；没有被裁剪掉的组件，则必须保留其定义的所有shell命令。

只要我们在工程中找个地方引用一下该文件的任意一个符号，就能保住该文件定义的全部shell命令，以及变量。例如，shell.c中有如下定义：

|  |
| --- |
| ADD\_TO\_ROUTINE\_SHELL(f,f,"写数据到内存,命令格式：f 起始地址……");  ADD\_TO\_ROUTINE\_SHELL(ver,djyos\_ver,"显示os的版本");  ADD\_TO\_ROUTINE\_SHELL(date,date,"显示或者设置日期");  ADD\_TO\_ROUTINE\_SHELL(d,d,"读取内存里的数据,命令格式:d 地址 ……");  ADD\_TO\_ROUTINE\_SHELL(time,time,"显示当前时间或者设置输入新时间"); |

只要引用一下其中任意一个符号就可以了。

|  |
| --- |
| abc = time |

看似很简单，然鹅却不是这么简单的。

我们在编码时，并不知道用户如何裁剪系统，哪些组件会被保留，哪些组件会被裁减掉，也就不知道哪些文件中的符号需要被引用。

谁知道呢？只有DIDE知道，引用符号这摊子事，只有让DIDE来干了。

方法：创建工程时，在“OS\_prjcfg\”路径中，创建一个keepshell.c文件，内容为：

|  |
| --- |
| //本文件由开发工具创建，勿修改。  void keep\_shell(void)  {  volatile void \*keep;  //DIDE扫描工程后，在此添加代码。  } |

注：keep\_shell函数并不会被调用。

在lds文件中，在text\_sysload\_xxxx段，添加如下代码：

|  |
| --- |
| KEEP(keepshell.o (.text .text.\* .rodata .rodata.\*)) |

打开或创建工程，或者修改组件配置后，根据工程裁剪情况，如果.a文件已存在，则打开.a文件，扫描里面的.o文件，逐个检查是否存在下列四个段中任意一个“ro\_shell\_cmd、ex\_shell\_cmd、ro\_shell\_data、ex\_shell\_data”，只要出现其中一个即停止扫描这个.o文件，在该段中任意找一个符号，在前文中keep\_shell函数中添加如下红色代码。

在每次编译.a文件后，亦执行此过程。

|  |
| --- |
| void keep\_shell(void)  {  volatile void \*keep;  extern void \*symbolname;  keep = (void \*)symbolname;  } |

注：不管符号本身是什么类型，都当做void \*解释。